Received: 9 July 2020 | Revised: 26 March 2021 | Accepted: 5 April 2021

DOI: 10.1002/spy2.164

Security and Privacy, Available online 5th May 2021, Article number e164
DOI:10.1002/spy2.164

RESEARCH ARTICLE

Android Permission Classifier: a deep learning algorithmic framework based on protection and threat levels

Moses Ashawa | **Sarah Morris**

Centre for Electronic Warefare Information and Cyber, Cranfield University, Shrivenham, UK

Correspondence
Moses Ashawa, Centre for Electronic Warefare Information and Cyber, Cranfield Univrsity, Shrivenham SN6 8LA, UK.
Email: m.ashawa@cranfield.ac.uk

Funding information
Petroleum Technology Development Fund

Recent works demonstrated that Android is the fastest growing mobile OS with the highest number of users worldwide. Android's popularity is facilitated by factors such as ease of use, open-source, and cheap to purchase compared to mobile OS like iOS. The widespread of Android has brought an exponential increase in the complexity and number of malicious applications targeting Android. Malware deploys different attack vectors to exploit Android vulnerability and attack the OS. One way to thwart malware attacks on Android is the use of Android security patches, antivirus software, and layer security. However, the fact that the permission request dynamic is different from other attack vectors, makes it difficult to identify which permission request is malicious or not especially when constructing permission request profiles for Android users. The aforementioned challenge is tackled by our research. This article proposed a framework called Android Permission Classifier for the classification of Android malware permission requests based on threat levels. This article is the first to classify Android permission based on their protection and threat levels. With the framework, out of the 113 permissions extracted, 23 were classified as more dangerous. Our model shows classification accuracy of 97% and an FPR value of 0.2% with high diversity capacity when compared with the performance of those of other similar existing methods.

KEYWORDS
Android permission request, feature diversity, machine learning algorithms, neural networks

1 | INTRODUCTION

Smartphones' increase in the market for business and personal purposes has brought about a corresponding increase in the number of malicious applications today. Malware attacks on mobile devices have become one of the most dreaded and precarious threats causing economic and information impairments. The study of Ashawa et al.[1,2] asserted that the increase and fast-growing rate at which mobile applications are been used in the 21st century has caused a proportionate growth in the number of cyber-attacks one of which is malware attacks on mobile and personal computers. Though, malicious applications target different mobile OS platforms, malware targeting Android has become a massive surge recently.

Smartphone market share recently published by the International Data Corporation (IDC) reported that Android OS shipment worldwide has slightly increased from 85.1% in 2018 to 86.7% in the second quarter of 2019 [3] and is the most popular mobile OS. The popularity of Android and its fast-growth shipment worldwide made the OS the major attack target by malware writers. This proliferation of different app from third parties makes application repacking and decompilation easier on the Android platform than could be done on the iOS platform due to Android's open-source.

Android popularity has made the most targeted mobile OS by mobile malware according to the recent research published by Kim et al.[4] Some of the factors which enhanced the popularity of Android OS including but not limited to open source, cheap to purchase, easy to use, and the launching of new models due to 5G inventions. Among other factors, open-source makes it easier for a lot of android developers to publish their applications on different Android market platforms without much critical security analysis and scrutiny, unlike iOS.

Different detections, classifications, and analysis techniques have been widely proposed by various researchers and security organizations across the world to curb the widespread of malware and to alleviate their attacks on Android devices. Some of the proposed techniques and strategies include dynamic detection,[5-12] static detection,[13-17] hybrid detection,[5,18-22] and memory forensics techniques, [23-31] respectively. The technique observes and monitors the application's behavior by focuses on knowing how the malware is connected and is interacting with mobile device resources at run time. The Static detection technique involves malware analysis without execution of the malware application but focuses on the evaluation of the signature and heuristic behaviour of the application using semantic properties. The hybrid technique is an amalgamation of both the static and dynamic technique features deployed for malware detection. It moderates at runtime portions of malware code to be inspected using static technique. Malware detection by the hybrid approach creates more stability in detection efficacy and accuracy by using advantages from both dynamic and static methodologies to provide more detection results and efficiency. The memory forensics analysis technique involves acquiring the device memory image which is infected or suspected using forensics memory acquisition tools and then analyzing the memory dump for forensic artifacts such as running processes, list of network connections, kernel modules, list of shared libraries, and rootkits. The technique is more reliable and efficient[32] when investigating sophisticated malware such as oligomorphic, polymorphic, and metamorphic.

Several types of research had been carried out on android malware detection, classification, and analysis. Our research is different from the rest because there has not been any research that focused on extracting and classifying Android malware permission requests based on their protection and threat levels using deep learning. While there are four permission attributes namely; normal permissions, dangerous permissions signature permissions, and signatureOrSystem permissions, our research focuses on the first two which are the normal and dangerous permissions. Android applications have many features including strings, opcode frequencies, API features, shared libraries, and so on which many previous studies have investigated. All of those features were generated by decoding the apk manifest file, decompiling the dex code, and disassembling the applications' shared libraries. Our framework on the other hand focused mainly on permission request features. Using APKtool,[33,34] we created permissions request feature vectors and refined them by decoding the Android Manifest file which was used to develop the algorithm for permission request classification. Using permission features extracted from manifest files, our classification framework maximizes the permission components and generated a feature vector representation. Each of the feature vectors was discretely inputted into the Android Permission Classifier (APC) framework. There are many Android malware sample dataset repositories such as DREDIN,[35] AndroZoo,[36] Genome project,[37] VirusShare.[38] Our framework performance was validated by series of experimental results using 6528 Android malware samples obtained from Ashawa and Sarah.[25] Effective permission extraction accuracy and classification based on the threat level of such permission request and the corresponding protection level was achieved by our model.

The following contributions were made by our research:

1. We proposed a novel Android malware permission request classification framework that employs only permission features that replicates Android malicious application attributes.
2. We proposed a neural network technique for the generation of malicious feature vectors with effectual classification and representation of malicious binary features with mutual qualities with benign or non-threatening Android applications.
3. We offered a permission classification method tagged APC using neural networks particularly deep learning. The method extracted all the permissions both malicious and benign applications requests and then separated malicious request from benign permissions.
4. We produced results that demonstrated our framework producing best validation performance at the minimum epoch when compared with other classifiers. To the best of our knowledge, there has not been any work that used deep learning to classify Android malware permissions request based on their threat and protection levels.

The rest of the article is structured as follows. Section 2 provides a review of the literature by discussing related works. Section 3 presents the methods and the algorithmic approach. Section 4 presents the proposed framework description. Section 5 presents our evaluation procedure and experimental results, assessing the diversity capacity of our framework. Section 6 makes some concluding remarks and discusses future work.

2 | BACKGROUND AND RELATED WORK

While numerous researches have been published on Android malware, a very limited amount of work has been done on malware protection and threat levels. In this section, related work on android malware similarity feature extraction and malware dynamic and static analysis were revisited. Then, we summarize and discuss permission requests and detection algorithms in the existing literature.

2.1 | Similarity feature extraction

Similarity exists between non-zero vectors which might help when determining feature standardization. The study of Schmidt et al.[39] in a multimodal deep learning method for android malware detection; using various feature sets proposed a framework that used a similarity-based system to extract and represent android malicious application features. The binary vectors and feature set were calculated and represented as similarity features by creating a match with their record and characteristics existence using a similarity-based feature vector generation algorithm. The extracted features represented the reflective characteristics of an Android-based application.

Using a machine learning algorithm, the study of McLaughin et al.[40] proposed an Android malware detection model that utilizes different Android Package Kit (APK) file features and permission requests information. Similar to the study of McLaughin et al.[40] the research of Saxe and Berlin[41] applied classification and clustering algorithms to obtain permission that malicious application request on Android-based platforms. The study performed by David and Netanyahu[42] used hardware and software features and components such as API calls, permission requests, and message intents to detect Android malicious applications. In addition, Kim[43] proposed a method whereby CFGs are used as features by utilizing keyword correlation gaps in FVG for Neural Support Vector Machine (SVM) classification. The SVM imposes its features to have a weight that is uniformly distributed.

An investigation into the leakage of sensitive information by Android malicious applications, a data flow exploration was performed by TaintDraoid dynamically McLaughlin[44]. Changes in the system' such as threads and processes were extracted and analyzed. Abnormalities identified showed different usage of system features such as CPU by malicious applications. The research of Wang et al.[45] performed a systematic review on different mobile detection techniques approaches. The result obtained in the research identified detection techniques with their respective strengths and weaknesses. Other studies such as Yerima[46] proposed a model for detecting mobile usage abnormalities.

2.2 | Permission requests and detection algorithms

Similarly, research[47] proposed a detection model where Convolutional Neural Network was used to detect malware on Android mobile devices. Sequences of opcode applications were used as features without alterations. Similarly, the study of Hao et al.[48] proposed a Deep Neural Network (DNN) framework for detecting malicious applications on Android-based devices. However, the method did not detect sophisticated and Zero-day malware that has similar attributes related to the benign application. Benign and malicious files were classified by the proposed work of Kim et al.[49] which uses Window dynamic API calls as feature vectors. The deep neural network-based model was believed to classify malicious files from benign files.

Related works that employed deep learning techniques include the research of TaeGuen et al.[52] which using a multimodal deep learning method, developed and Android detection model using various features. The model learning strategy was based on a neural network. Using seven features namely String, opcode, API, shared library function, permission features, component, and environmental features; the model merged all the features into one feature and fed them into the classifier for malware detection. Their research did not in any given consideration on dangerous permissions request classification. The study of Jei et al.[51] using Convolutional neural network proposed a detection technique for Android malware. The detection model used opcode features of android applications without modification.

Investigating the permissions with the risk that malicious applications use to explore Android resources, the study of Jha et al.[50] extracted permissions requests by Android applications using Random Forest. However, less relevant features that are inherent to an application were not explored. Though several previous works were carried out on identifying malicious applications using neural networks, none has investigated extracting and classifying permissions requests that android applications request using deep learning. In addition, none of the previous works consider the threat level that dangerous permission requests posed on the protection level of Android mobile devices. The study of Pei et al.[51] carried a systematic review on Android malware detection techniques. In the comparative results, the research identified the algorithms deployed in each of the techniques and their detection accuracy with their identified strengths and limitations, respectively.

The research intensively explored all the well-known detection techniques such as dynamic, static, hybrid, content-based, and emulation-based detection. However, they did not cover deep learning techniques for Android malware detection. What distinguished our work from the previous work is that though many of the previous research used several features and did not focus on classifying permission requests based on their threat and protection levels. Though several previous works were carried out, none has investigated the threat level that dangerous permission requests posed on the protection level of Android mobile devices. This formed part of the novelty of our research.

3 | MATERIALS AND METHODS

3.1 | Malware data set

The Android malware dataset used for this research was obtained from Contagio[31] and DREBIN[46] Android malware dataset repository. Using an online VirusTotal scanner, we scanned to ascertain the benignity and maliciousness of the sample files. After scanning, the APK file was unzipped to extract the file contents of the dataset folder. The dataset consisted of 5560 malicious applications and 9476 benign application samples. The whole dataset relation consisted of 15 035 instances in an APK file format.

3.2 | APK file decompilation

To obtain the resources contained in the raw dataset, the APK file was decompiled using Apktool[47] embedded in the newest version of the Androguard.[48] The decompiled APK contains lib, res, original assets, and the Android Manifest file application resources. The output of the decompiled APK generated Smali and Java class folders holding the resource files. Features such as the manifest permissions, API call signatures, intents, and command signatures were obtained to form feature sets and input vectors to the network. Other features such as shared lib and libdata were also extracted after disassembling the source code of the .so file. The manifest permissions were extracted from the AndroidManifest.xml file extension while the API call signatures were extracted from Android class files. Using class conditional probabilities with minimum values, a total of 215 attribute partitions were extracted from pr_partition_0. The attributes extracted formed the general feature set. Specifically, the attributes contain 113 manifest permissions, 72 API call signatures, 23 intents, 6 command signatures, and binary attributes for (Benign = B, Malware = S [1]). In summary, our proposed classifier uses the following features: permission feature, API call feature, command feature, and intent feature, respectively.

3.2.1 | Permission features

Permission manifest is one of the essential access control security mechanisms that Android uses. This access control mechanism (permission) places a constraint on the operational performance of any Android process. Permissions requested by applications provide relevant information regarding the application's behavior. Permissions features are usually defined or declared in the manifest file of the Android APK. During application installation, the manifest.xml file [49-51] holds significant information about an application. When permission is granted, the application can be able to use and interact with the device- protected features. Android permissions are classified into four distinct threat levels namely, "normal permission, dangerous permissions, signature permission, and signature/System permission."[52] This study grouped both protection levels and manifest permissions as permissions features for the framework.

3.2.2 | API call features

Android architecture is designed in such a way that the API of the system framework interacts with the Android system. Android API is made of two major components namely: classes, and package.[51] Malicious applications often invoke API. API invocational provides information on the application behavior. Reverse engineering of the Android APK extracted all the API features in the dataset used for this research.

3.2.3 | Command signature features

These are the command instruction sets for signing the Android label and activity class, respectively. Command signatures are used to sign requested permissions that applications request. In the case of benign applications, command signature signs applications whose signature certificate is the same as that of the permissions declared. When there is a match of certificates, Android automatically grants access to the application, without notifying the user or seeking his approval to grant access. Malicious applications that have the same signature as the declared permissions in the device protection level can easily have access to the system resources in the background. While permission request features of a malicious file are harmful to system information, command features are to the system's data and hardware.

3.2.4 | Intent features

An Intent is theoretical or abstract decryption of an action or activity to be implemented or performed. Android uses intent to launch and broadcast processes to any component of interest. Intents communicate with different services in the background; thus, facilitating runtime binding of different application codes. Android uses intent features to launch activities. Intent contains two primary attributes, action and data. Other secondary attributes such as the category of the action to be performed on the data, component class to be used and, extras formed the metadata.

3.3 | Feature selection and database generation

To improve the classifier's performance efficiency and reduce its computational cost, we reduced the number of the model's input variables. The selected input variables in this research were those whose correlation with the target variables was strongest. To achieve this, feature ranking approach[53] was applied as a preliminary phase in determining the application characteristics and input variables reduction. Various android features such as the API call signatures, intents, manifest permissions, and command signatures were extracted from the apk as explained in subsection 3.2. These features formed the general feature vectors. Features extracted (see Section 3.2) formed the database of the feature vectors. In generating, the database for the feature vectors, the extracted features were classified into two different features based on their class pattern. To normalize the feature value, the mix-max scaling approach[54] was applied by scaling feature values between the intervals of 0 to 1. Features with no value attributed to being scaled as zero (0) while those with values attributed to were scaled as one (1), respectively. Using the K-means clustering method, eighty (8) iterations were performed within-cluster sum of squared errors. C0 and C1 are the cluster labels for classes1 and 2 with counts 66 and 34, respectively. The class attribute has 66.0 weight in C0 and 34.0 weight in C1(as shown in Table 1). These formed the cluster centroids for the feature vector attributes of the dataset attributes distribution (as shown in Table 2).

A total of eight iterations was performed within-cluster sum of squared errors using the K-means clustering method. C0 and C1 are the cluster labels for classes 1 and 2 with counts 66 and 34, respectively. The class attribute has 66.0 weight in C0 and 34.0 weight in C1. These formed the cluster centroids where permission attributes are distributed accordingly. Cluster attributes logical table contain the attribute class and the logical values (as shown in Table 3) while the decision list (as shown in Table 4) shows the associated rules formed from the attribute logical tables, respectively.

All the feature categories were then combined to form a unified set of vectors which were used as the input vector. The input vector was converted into Comma Separated Values (CSV). The reason for converting the input vector into CVS format was to make it friendly with tools that find it difficult to accept datasets without a field separator. All the four feature categories were generated using K-means. Then combined to form a unified set of vectors which were used as the input vector. The input vector was converted into CSV. The reason for converting the input vector into CVS format was

TABLE 1 Clusters containing feature vector binary attributes

Clusters	Binary features
Cluster 0 (c0): S	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,1,0,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0, 0,0,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,S
Cluster 1 (c1): B	1,1,1,1,1,1,0,1,1,1,1,1,0,0,0,1,1,1,1,1,0,1,0,0,1,0,0,0,1,1,0,0,1,1,1,0,0,1,0,0,0,0, 0,0,1,0,1,0,1,0,0,1,1,0,1,0,0,1,1,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,1,0,0,1,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0,0,0,0,1,0,0,1,1,1,0,0,0,B

TABLE 2 Data set attributes distribution

Statistical attributes	Distinct values
Minimum	0
Maximum	1
Mean	0.426
StdDev	0.495
Class pattern	8624
Benign	[0, 0.071]
Class pattern	6412
malicious	[0.929, 1]

Note: Statistical attributes: The qualitative characteristics of malware dataset, Distinct values: These are calculated values held by each statistical attribute, Minimum: 0 and 1 are the minimum and maximum binary values contained in each cluster (Cluster 0, Cluster 1), respectively, Mean: statistical mean of the dataset distribution, StdDev: SD,

to make it friendly with tools that find it difficult to accept datasets without a field separator. The algorithm for feature vector generation (as shown in Algorithm 1).

Algorithm 1. Generation of combined feature vectors from the binary feature vector database

```
Input: Android applications' feature set contained in the Database.
Output: Combined feature vectors.
1: decompile APK // to access all the APK components.
2: set_partition_index_feature_vector⟵[1| 0| 0| 1| 0| … 0| 1].
3: centroids⟵ K_means
4: for index⟵0 // setting the feature vector index
5: for $\forall\ \mathscr{F}s \in \beta\mathscr{F}V\_db$ do while
6: if $\mathscr{F}s \in \mathscr{F}V$_Android_app
7: then
8: combined_feature_vectors[index] ⟵1
9: return combined_feature_vectors
10: exit
```

The feature vector information extracted from the dataset form the malicious feature database which consists of strings, permissions, components, and other environmental features. The structure of the database composed of different

TABLE 3 Attributes logical table

Attribute	Logical values
@attribute a0	{false,true}
@attribute a1	{false,true}
@attribute a2	{false,true}
@attribute a3	{false,true}
@attribute a4	{false,true}
@attribute a5	{false,true}
@attribute a6	{false,true}
@attribute a7	{false,true}
@attribute a8	{false,true}
@attribute a9	{false,true}
@attribute class	{c0,c1}

components where all the features were pruned and classified. All the string features with their respect hashes and sizes, permission features are in <uses-permission> and < permission> tag. All:<provider>, <filter>, <activity>, <intent>, <grant-uri permission>, <service>, and < receiver> formed the general component of the feature database were all the distinct features are extracted and classified. Environmental components: <library> and < uses-sdk>. Opcode and API methods: send_message, socket, and invoke. Details of the feature vector set information extracted by the algorithm which contained intent (as shown in Table 5), commands signature (as shown in Table 6), permission (as shown in Table 7), and API call signature (as shown in Table 8) formed the major feature categories of the combined database, respectively.

3.4 | Feature classification

The approach used for the feature classification process in this research is the backpropagation mechanism.[55] Back-propagation constitutes the central learning hub for producing an accurate classification outcome. It is the information transmission with the error which neural network generates when trying to make a prediction or classification about a given dataset. This is composed of the iteration and learning layers (input, hidden, and output layer) with distinctive random initialization weights and biases. Our model applied information gain approach[56] as feature classification methodology. The *IG* processes the amount of information a feature X provides about the class. Let X be a feature in a given Android application sample contain x_i possible outcomes, the *IG* is represented as follows:

$$IG = H(X) - H(x/y) \tag{1}$$

where $H(x)$ is the measure of randomness in the processed information which forms the entropy of feature X, $H(x/y)$ is the entropy after observing Y features. Given the probability density function (PDF) $p(x)$ for random variable X, the $H(X)$ of X feature defined as:

$$H(X) = -\sum p(x)\log_2(p(x)) \tag{2}$$

Similarly, the $H(X)$ of X comparative to feature Y is defined as:

$$H\left({}^{x}\!/_{y}\right) = -\sum p\left({}^{x}\!/_{y}\right)\log_2\left(p\left({}^{x}\!/_{y}\right)\right) \tag{3}$$

where $p(x/y)$ is the conditional probability. High diminution of the entropy in X increases the significance of its features. *IG* decreases at the apex of PDF when the conditional probability in the sample pattern of both malicious and benign features increases; while the entropy in X decreases as PDF decreases, respectively. We assumed that there was no probabilistic failure that existed at any time (t) during the attribute's distribution and clustering.

Table 4 Decision list using associated rules

Rule	Decision List
RULE 0:	c0: = not(a3), not(a2), a6, a0
RULE 1:	c1: = not(a9), a2, not(a4), not(a1), a5
RULE 2:	c0: = a3, a4, a6, not(a9), not(a5)
RULE 3:	c1: = a1, a6, a0, not(a7), a3, a9, not(a5), a8
RULE 4:	c0: = not(a5)
RULE 5:	c1: = a4, not(a3), a2, not(a8), a9, not(a0), a5
RULE 6:	c0: = not(a2), a3, a7, not(a4), not(a6), a1, a0, a8, a5
RULE 7:	c1: = not(a8), a5, a9
RULE 8:	c0: = a6, not(a3), not(a4)
RULE 9:	c1: = a5, not(a6), not(a2), not(a9), not(a7), a4
RULE 10:	c0: = a5, not(a4), not(a2)
RULE 11:	c1: = not(a2), not(a6), a7, a8, a3, not(a0), a5, not(a9)
RULE 12:	c0: = a9, not(a3), a1, a0, a5
RULE 13:	c1: = not(a9), a6, a7, a4, a2, a3, not(a1), not(a8), a0
RULE 14:	c0: = a6, a0, a5, a8, a4, a7, a3
RULE 15:	c1: = not(a9), a0, not(a8), a6, a5, a4, a1
RULE 16:	c0: = a1, a3, a5, a9, a2
RULE 17:	c1: = not(a4)
RULE 18:	c0: = a2
RULE 19:	c1: = not(a0), not(a6), a3, a8, a7, a4, a9
RULE 20:	c0: = not(a0), a8, not(a2), not(a9), a5

On the other hand, if a probability failure occurs at any given survival; the risk function $h(t)$ will occur by forming an integral and instantaneous ratio. The hazard or risk function $h(t)$ was defined as follows:

$$H(x) = \int_{\infty}^{x} h(t)dt \tag{4}$$

where ∞ is the range of time t from 0 to ∞. Our research assumed that all the features in the dataset survive the past time and hazard function. Thus, the survival function was defined as:

$$s(t) = p_r(T > t) = 1 - F(t) \tag{5}$$

where $S(t)$ is the survival function at time t, T is the random variable, P_r is the probability that some feature survived, and F is the event density. Using three different distributions: Weibull, normal, and Birnbaum–Saunders, we demonstrated that at each intersection of either of the distributions, as F increases, the corresponding value of $S(t)$ decreases, respectively. This provides a guide in studying the binary vectors in the dataset during feature representation.

3.5 | Binary feature representation

In this subsection, we described the permission classification problem of our proposed classifier. Our binary feature vectors were obtained by performing eight iterations within-cluster sum of squared errors using the K-means clustering method. Let m represent a set of given G programs given as $G = \{g_1 \ldots\ g_m\}$ with $g_i \in G$. Let the class label (C_i) and the

TABLE 5 Intent feature vectors

Feature	Category
android.intent.action.BOOT_COMPLETED	Intent
android.intent.action.PACKAGE_REPLACED	Intent
android.intent.action.SEND_MULTIPLE	Intent
android.intent.action.TIME_SET	Intent
android.intent.action.PACKAGE_REMOVED	Intent
android.intent.action.TIMEZONE_CHANGED	Intent
android.intent.action.ACTION_POWER_DISCONNECTED	Intent
android.intent.action.PACKAGE_ADDED	Intent
android.intent.action.ACTION_SHUTDOWN	Intent
android.intent.action.PACKAGE_DATA_CLEARED	Intent
android.intent.action.PACKAGE_CHANGED	Intent
android.intent.action.NEW_OUTGOING_CALL	Intent
android.intent.action.SENDTO	Intent
android.intent.action.CALL	Intent
android.intent.action.SCREEN_ON	Intent
android.intent.action.BATTERY_OKAY	Intent
android. Intent.action.PACKAGE_RESTARTED	Intent
android.intent.action.CALL_BUTTON	Intent
android.intent.action.SCREEN_OFF	Intent
intent.action.RUN	Intent
android.intent.action.SET_WALLPAPER	Intent
android.intent.action.ACTION_POWER_CONNECTED	Intent
android.intent.action.BATTERY_LOW	Intent

TABLE 6 Commands signature feature vectors

Feature	Category
mount	Commands signature
chmod	Commands signature
remount	Commands signature
chown	Commands signature
/system/bin	Commands signature
/system/app	Commands signature

feature vector (f_i) matched as $g_i \in G$ for every $g_i \in G$. We defined our class label and feature vector as follows:

$$C_i \in \{c_m, c_b, c_k\} \tag{6}$$

$$f_i = \{f_1, f_2, \dots, f_n\} \tag{7}$$

where C_m, C_b, C_k represents class label for the malicious request, class label for the benign permission request, and class label for an unknown permission request respectably. Let $R = \{r_{1,}\ r_{2,\ \dots,}\ r_{n,}\}$ represent n classifiers set that label permission requests from the input apk file as either benign or malicious. Let $r_i = \{a_i, C_i, T_i\}$, where T_i is the classifiers' computational time, a_i is the classification accuracy, and C_i is the function that labels and maps the feature vector f to c and a_i, respectively, upon the training set represented as $C\text{i}(f): f \longrightarrow c$.

TABLE 7 Permission feature vectors

Feature	Category	Feature	Category
SEND_SMS	Manifest Permission	WRITE_SECURE_SETTINGS	Manifest Permission
READ_PHONE_STATE	Manifest Permission	DUMP	Manifest Permission
GET_ACCOUNTS	Manifest Permission	BATTERY_STATS	Manifest Permission
RECEIVE_SMS	Manifest Permission	ACCESS_COARSE_LOCATION	Manifest Permission
READ_SMS	Manifest Permission	SET_TIME	Manifest Permission
USE_CREDENTIALS	Manifest Permission	WRITE_SOCIAL_STREAM	Manifest Permission
MANAGE_ACCOUNTS	Manifest Permission	WRITE_SETTINGS	Manifest Permission
WRITE_SMS	Manifest Permission	REBOOT	Manifest Permission
READ_SYNC_SETTINGS	Manifest Permission	BLUETOOTH_ADMIN	Manifest Permission
AUTHENTICATE_ACCOUNTS	Manifest Permission	BIND_DEVICE_ADMIN	Manifest Permission
WRITE_HISTORY_BOOKMARKS	Manifest Permission	WRITE_GSERVICES	Manifest Permission
INSTALL_PACKAGES	Manifest Permission	KILL_BACKGROUND_PROCESSES	Manifest Permission
CAMERA	Manifest Permission	STATUS_BAR	Manifest Permission
WRITE_SYNC_SETTINGS	Manifest Permission	PERSISTENT_ACTIVITY	Manifest Permission
READ_HISTORY_BOOKMARKS	Manifest Permission	CHANGE_NETWORK_STATE	Manifest Permission
INTERNET	Manifest Permission	RECEIVE_MMS	Manifest Permission
RECORD_AUDIO	Manifest Permission	SET_TIME_ZONE	Manifest Permission
NFC	Manifest Permission	CONTROL_LOCATION_UPDATES	Manifest Permission
ACCESS_LOCATION_EXTRA_COMMANDS	Manifest Permission	BROADCAST_WAP_PUSH	Manifest Permission
WRITE_APN_SETTINGS	Manifest Permission	BIND_ACCESSIBILITY_SERVICE	Manifest Permission
BIND_REMOTEVIEWS	Manifest Permission	ADD_VOICEMAIL	Manifest Permission
READ_PROFILE	Manifest Permission	CALL_PHONE	Manifest Permission
MODIFY_AUDIO_SETTINGS	Manifest Permission	BIND_APPWIDGET	Manifest Permission
READ_SYNC_STATS	Manifest Permission	FLASHLIGHT	Manifest Permission
BROADCAST_STICKY	Manifest Permission	READ_LOGS	Manifest Permission
WAKE_LOCK	Manifest Permission	SET_PROCESS_LIMIT	Manifest Permission
RECEIVE_BOOT_COMPLETED	Manifest Permission	MOUNT_UNMOUNT_FILESYSTEMS	Manifest Permission
RESTART_PACKAGES	Manifest Permission	BIND_TEXT_SERVICE	Manifest Permission

TABLE 7 (Continued)

Feature	Category	Feature	Category
BLUETOOTH	Manifest Permission	INSTALL_LOCATION_PROVIDER	Manifest Permission
READ_CALENDAR	Manifest Permission	SYSTEM_ALERT_WINDOW	Manifest Permission
READ_CALL_LOG	Manifest Permission	MOUNT_FORMAT_FILESYSTEMS	Manifest Permission
SUBSCRIBED_FEEDS_WRITE	Manifest Permission	CHANGE_CONFIGURATION	Manifest Permission
READ_EXTERNAL_STORAGE	Manifest Permission	CLEAR_APP_USER_DATA	Manifest Permission
VIBRATE	Manifest Permission	CHANGE_WIFI_STATE	Manifest Permission
ACCESS_NETWORK_STATE	Manifest Permission	READ_FRAME_BUFFER	Manifest Permission
SUBSCRIBED_FEEDS_READ	Manifest Permission	ACCESS_SURFACE_FLINGER	Manifest Permission
CHANGE_WIFI_MULTICAST_STATE	Manifest Permission	BROADCAST_SMS	Manifest Permission
WRITE_CALENDAR	Manifest Permission	EXPAND_STATUS_BAR	Manifest Permission
MASTER_CLEAR	Manifest Permission	INTERNAL_SYSTEM_WINDOW	Manifest Permission
UPDATE_DEVICE_STATS	Manifest Permission	SET_ACTIVITY_WATCHER	Manifest Permission
WRITE_CALL_LOG	Manifest Permission	WRITE_CONTACTS	Manifest Permission
DELETE_PACKAGES	Manifest Permission	BIND_VPN_SERVICE	Manifest Permission
GET_TASKS	Manifest Permission	DISABLE_KEYGUARD	Manifest Permission
GLOBAL_SEARCH	Manifest Permission	ACCESS_MOCK_LOCATION	Manifest Permission
DELETE_CACHE_FILES	Manifest Permission	GET_PACKAGE_SIZE	Manifest Permission
WRITE_USER_DICTIONARY	Manifest Permission	MODIFY_PHONE_STATE	Manifest Permission
REORDER_TASKS	Manifest Permission	CHANGE_COMPONENT_ENABLED_STATE	Manifest Permission
WRITE_PROFILE	Manifest Permission	CLEAR_APP_CACHE	Manifest Permission
SET_WALLPAPER	Manifest Permission	SET_ORIENTATION	Manifest Permission
FEATURESBIND_INPUT_METHOD	Manifest Permission	READ_CONTACTS	Manifest Permission
READ_SOCIAL_STREAM	Manifest Permission	DEVICE_POWER	Manifest Permission
READ_USER_DICTIONARY	Manifest Permission	HARDWARE_TEST	Manifest Permission
PROCESS_OUTGOING_CALLS	Manifest Permission	ACCESS_WIFI_STATE	Manifest Permission
CALL_PRIVILEGED	Manifest Permission	WRITE_EXTERNAL_STORAGE	Manifest Permission
BIND_WALLPAPER	Manifest Permission	ACCESS_FINE_LOCATION	Manifest Permission
RECEIVE_WAP_PUSH	Manifest Permission	SET_WALLPAPER_HINTS	Manifest Permission
		SET_PREFERRED_APPLICATIONS	Manifest Permission

TABLE 8 API call signature feature vectors

Feature	Category	Feature	Category
onServiceConnected	API call signature	createSubprocess	API call signature
bindService	API call signature	URLClassLoader	API call signature
attachInterface	API call signature	abortBroadcast	API call signature
ServiceConnection	API call signature	TelephonyManager.getDeviceId	API call signature
android.os.Binder	API call signature	getCallingPid	API call signature
Ljava.lang.Class.getCanonicalName	API call signature	Ljava.lang.Class.getPackage	API call signature
Ljava.lang.Class.getMethods	API call signature	Ljava.lang.Class.getDeclaredClasses	API call signature
Ljava.lang.Class.cast	API call signature	PathClassLoader	API call signature
Ljava.net.URLDecoder	API call signature	TelephonyManager.getSimSerialNumber	API call signature
android.content.pm.Signature	API call signature	Runtime.load	API call signature
android.telephony.SmsManager	API call signature	TelephonyManager.getCallState	API call signature
getBinder	API call signature	TelephonyManager.getSimCountryIso	API call signature
ClassLoader	API call signature	sendMultipartTextMessage	API call signature
Landroid.content.Context.registerReceiver	API call signature	PackageInstaller	API call signature
Ljava.lang.Class.getField	API call signature	sendDataMessage	API call signature
Landroid.content.Context.unregisterReceiver	API call signature	HttpPost.init	API call signature
Ljava.lang.Class.getDeclaredField	API call signature	Ljava.lang.Class.getClasses	API call signature
getCallingUid	API call signature	TelephonyManager.isNetworkRoaming	API call signature
Ljavax.crypto.spec.SecretKeySpec	API call signature	HttpUriRequest	API call signature
android.content.pm.PackageInfo	API call signature	divideMessage	API call signature
KeySpec	API call signature	Runtime.exec	API call signature
TelephonyManager.getLine1Number	API call signature	TelephonyManager.getNetworkOperator	API call signature
DexClassLoader	API call signature	MessengerService	API call signature
HttpGet.init	API call signature	IRemoteService	API call signature
SecretKey	API call signature	Set_Alarm	API call signature
Ljava.lang.Class.getMethod	API call signature	Account_Manager	API call signature
System.loadLibrary	API call signature	TelephonyManager.getSimOperator	API call signature
android.intent.action.SEND	API call signature	onBind	API call signature
Ljavax.crypto.Cipher	API call signature	Process.start	API call signature
android.telephony.gsm.SmsManager	API call signature	Context.bindService	API call signature
TelephonyManager.getSubscriberId	API call signature	ProcessBuilder	API call signature
Runtime.getRuntime	API call signature	Ljava.lang.Class.getResource	API call signature
Ljava.lang.Object.getClass	API call signature	defineClass	API call signature
Ljava.lang.Class.forName	API call signature	findClass	API call signature
Binder	API call signature	Runtime.loadLibrary	API call signature
android.os.IBinder	API call signature	transact	API call signature

Our model computational time (TR) and the classification rate of accuracy (AR) is defined by the following equations, respectively as:

$$TR = \sum_{i=1}^{n} T(r_i) \mid r_i \in R \tag{8}$$

$$AR = \sum_{i=1}^{n} a(r_i) \mid r_i \in R \tag{9}$$

To prevent the classifiers from going beyond the threshold level (φ) given that $AR \leq \varphi$ and all the n classifiers in our model must not exceed the minimum classifiers' number ($\mathbb{E}$), we defined the minimum function in the system operates as follows:

$$\min_{n} = \sum_{i=1}^{n} r_i \leq E \tag{10}$$

Given a condition that each permission requested by the application is labeled as either malicious or benign as $g_i \in \{C_m, C_b\}$, $\forall g_i \in G$, $1 \leq i \leq m$; where C_m represents class label for the malicious permission request, C_b represents class label for the benign permission request, respectively. We included the classifiers' number $\mathbb{E}$ to help determine the effectiveness of our classifier. It also helps comparison with different machine learning classifiers in case there is a disparity in classification performance between them. Each binary number represents a cluster attribute in the decision list. Using 2-fold cross-validation, we constructed attributes rules with each cluster in the decision list. Concerning the feature vectors, each attribute can either be a false or a true and must contain each value representation in both clusters (cluster 0 and cluster 1). All the features were spread over pr_partion_0 which shows the distribution value for each attribute in a density-based clustering pattern. The cluster attributes from attribute a0 to attribute a9 were formed using association rules[57] for feature mining attributes logical values (see Algorithm 2).

Algorithm 2. Classification of permission feature vector from the database

```
Input: Android applications' manifest xml file
Output: Android permission features
1: all permissions ⟵ $\forall \delta$ // generating permission information
2: for 0 to Max_iteration do
3: if all perm==$\delta$
4: then for 0 to total perm do
5: malicious perm⟵select feature (0, 1) // padded zeros and leading zeros
6: newperm ⟵newperm $\bigcup$\{malicious\}
7: else
8: for new.total () < totalperm do
9: select K, L$\in$ (0, |allpermissions|)
10: malicious = K_crossover (allpermissions[k], allpermissions[l])
11: for maliciousperm$\in$ newperm do
12: for benignperm$_k$ $\in$ newperm do
13: if $\exists$ maliciousperml maliciousperm$_l$ $\in$ newperm $\bigwedge$ k$\neq$$\boldsymbol{l}$.maliciousperm$_k$ $\succ$ maliciousperm$_l$
14: then
15: newperm ⟵newperm\ maliciousperm$_k$
16: if newperm $\sqsubset$ $\forall$$\boldsymbol{maliciousperm}$ then
17: permission_feature_vector [index] ⟵1
18: else
19: permission_feature_vector [index] ⟵$\mathbf{0}$
20: return android_permission_feature_vectors
21: exit
```

4 | FRAMEWORK DESCRIPTION

This subsection of the article describes the neural network approach applied in our proposed framework known as APC. The principle used in developing our classification model is the principle of convolutional neural networks for recognizing patterns in datasets.[58,59] We defined our neural network system $\mathbb{S} \in \{1, 2, 3\}$ to consist of $\boldsymbol{l}$ number of layers defined as $\boldsymbol{l}^{\mathbb{S}} = \{1,2, \ldots, \mathrm{n}\}$, respectively. In our framework, the vector input x that enters the initial network was represented as$\mathscr{x}^{\mathbb{S}}$. From linear transformation, the output from the initial network was transformed from a matrix to another vector ($\mathscr{Z}$) into the second layer to form the incoming vector represented as $\mathscr{Z}^{(\mathscr{x}^{\mathbb{S}})}$. Where $\mathscr{Z}$ is the transformation function. From partial differentiation, $y = f\ (\mathscr{x})$ where y is the output of the function. The rectified linear unit (ReLu) as the activation function applied in our neural network model is defined as:

$$y = f(x) = \begin{cases} 0\, for & x < 0 \\ x\, for & x \geq 0 \end{cases} \tag{11}$$

The transformation z over the input vector x, weights W and biases b on the above function is defined as:

$$z = f(x, w, b) \tag{12}$$

$$y = g(x) \tag{13}$$

To design a classification model that captures multidimensional data inputs with weights and biases, we applied max-pooling operation after convolution with Relu function as follows:

$$z = \tanh(f(x_1, w_1, b_1)) \tag{14}$$

$$y = \mathrm{Re}lu(f(z, w_2, b_2)) \tag{15}$$

$$\mathrm{Re}lu(x) = \max(0, z) \tag{16}$$

where ReLu is the activation function of the network, x is the input vector, w is the network weight and b are the biases in the network, respectively. The output, weights, and biases from the network layer $l^{\mathbb{S}}$while applying the ReLu function (activation function) in the initial network using backpropagation was defined by the following equations:

$$y(l^s + 1) = f(Z(l^s + 1)) \tag{17}$$

$$z(l^s + 1) = w(l^s + 1).y(l^s) + b(l^s + 1) \tag{18}$$

where W are the network weights, b are the network biases, $y(\mathrm{l}^{\mathbb{S}})$ output vector, and f is the network activation function, respectively. Based on the working principle illustrated in the transformation operation shown in the figure above, the output in the initial network layer was been transformed and became an input vector in the final network of our neural network model as follows:

$$\mathrm{z}'(\langle i' + 1\rangle) = w'(\langle i' + 1\rangle).y'(l^s) + b'^{(l^s+1)} \tag{19}$$

To help predict the probability of the output between 0 and 1 range, the squashing function (σ) was applied at the last layer of the network. This is to determine permission is malicious (1) or benign (0) in our framework. Given the quashing function σ, the number of layer n in the neural network, the output vector of the final layer y', the input vector x in the network; The transformation at the last layer of the network was defined as follows:

$$y'(n_x) = \sigma(z'^{(n_x)}) \tag{20}$$

where σ is the applied squashing function, n is the number of layers in the neural network, y' is the output vector of the final layer, x is the input vector in the network, and Z is the matrix vector, respectively. The overall system architecture (as shown in Figure 1) illustrates the various processes performed from dataset collection to binary feature vector generation.

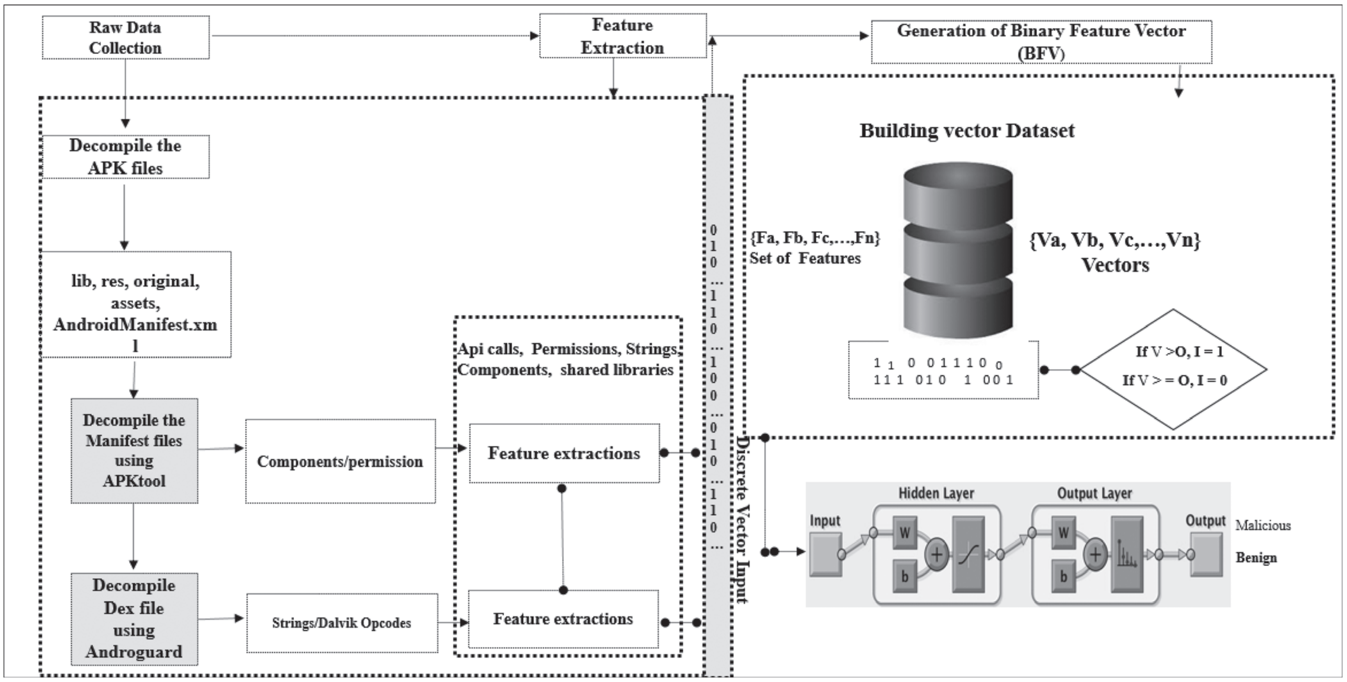

Raw Data Collection
Feature Extraction
Generation of Binary Feature Vector (BFV)
Decompile the APK files
lib, res, original, assets, AndroidManifest.xml
Decompile the Manifest files using APKtool
Components/permission
Decompile Dex file using Androguard
Strings/Dalvik Opcodes
Api calls, Permissions, Strings, Components, shared libraries
Feature extractions
Feature extractions
Discrete Vector Input
Building vector Dataset
{Fa, Fb, Fc,…,Fn} Set of Features
{Va, Vb, Vc,…,Vn} Vectors
If V >O, I = 1
If V > = O, I = 0
Input
Hidden Layer
Output Layer
Output
Malicious
Benign

FIGURE 1 The overall system architecture

4.1 | Network overfitting

When developing a classification model using a neural network, network overfitting characterizes the basic drawback in many models. To avoid overfitting occurrence, our research applied Bayesian regularization (BR) technique.[60,61] The choice to choose the Bayesian regularization over other algorithms such as Levenberg–Marquardt[48] is its capability to show potential convoluted relationships between variables in the distribution of a dataset. Therefore, providing robustness in the model. Unlike Dropout regulation[62] that skips some unit randomly during training of the network, Bayesian regularization technique skips the model from bias definition,[63] thus, making the model totally reliant on biases and weights associated with the network units. Using BR in the APC model provides the framework with enhanced generation performance and achieved the best network validation performance.

The overall problem definition of the Bayesian regularization training to overcome network overfitting is defined in the equation as follows:

$$y(t) = f(x(t-1), \dots, x(t-d), y(t-1), \dots, y(t-d)) \tag{21}$$

where x is the random variable taken definite values between 0 and 1 in the dataset of X independent variables, d is the number of delays in the network, t is the target variable, f is the nonlinear function of x variables, and $y(t)$ is the output of the target value. Our dataset was trained by introducing a linear function w to minimalize the error in t. The expectation (E) of the nonlinear function f is defined as:

$$E[f] = \int p(x).f(x)dx \tag{22}$$

We defined the average expected loss L when a malicious permission is wrongly classified as benign as follows:

$$E[L] = \iint L(t, y(x))p(x,t)dxdt \tag{23}$$

where $L(t, y(x)) = (y(x) - t)^2$ and $p(x, t)$ are normal distribution parameters. The MSE over the parameterized function is the model cost function. The neural network operation (as shown in Figure 2) schematically shows the operation function of the proposed classifier.

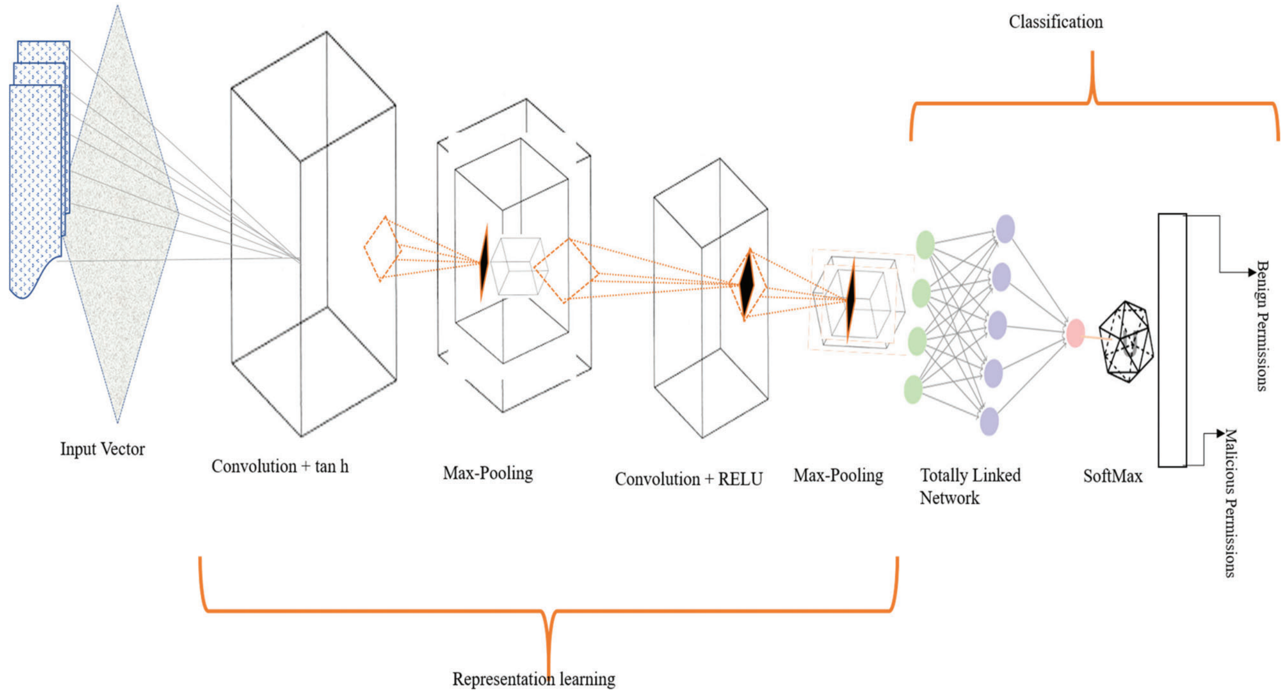


FIGURE 2 An illustration of the neural network working operation function of the proposed classifier

4.2 | Regularization of feature diversity

Feature redundancy (FRc) is one of the major factors that enhance errors in classification and detection frameworks. Most conventional classification algorithms are limited to inaccurate discrimination of features related to malware. This is due to the daily emergence of different malware variants with unique attributes. To minimize classification errors that could occur because of FRc, diversity regularization technique [64] was applied in our framework. During this process, we discarded all the features with self-correlation and only those whose correlation values are above the diversity threshold τ were considered. The diversity threshold provides a trade-off parameter to ensure that feature vectors in the malware dataset are adequately differentiated. Feature sharing across several high-level attributes is ensured. Consider a vector space $V = \{f_1 f_2, \dots, f_n\}$, were f_s are feature vectors. The similarity that exists between two feature vectors say, f_1 and f_2 is determined by the correlation that occurs between f_1 & f_2.

During the training process, SIM_c is set to take definite values within the range of −1 to 1. The SIM_c value is negative (−1) when f_1 & f_2 have opposite direction in the feature vector subspaceV. This does not connote the vectors in this direction are negative feature vectors. SIM_c Value is set to zero (0) if f_1 & f_2 in the vector space are in an orthogonal projection direction. For instance, when f_1 & f_2 are in orthogonal projection, we defined it in a bilinear form as follows: $<f_1 \& f_2> = 0$ for $i \neq j$. The regularization cost for feature vector diversity abbreviated as (D_rC) in the APC neural network layer l^{th} with connecting feature networks given n number of neurons in l layer, and the binary mask variable $m_{i,j}$,.

To determine the earliest time that features classification failure may occur, the threshold parameter τ was taken as a hyperparameter assuming values between 0 and 1. During the experiment, we avoided setting the hyperparameter to zero (0) to avoid feature orthogonality. This was considered due to feature sharing abilities that exist in android based applications. To enforce feature diversity during training, the diversity factor λ in the experiments was performed at 1, 5, 15, and 30, respectively, using 100 iterations. The threshold parameter was performed at τ=0.1, 0.2, and 0.3, respectively. The regularization cost for feature vector diversity abbreviated as (D_rC) in the APC neural network layer l^{th} with connecting feature networks given n number of neurons in l layer, and the binary mask variable $m_{i,j}$, edged with the SIM_c and the features in the vector space is defined as:

$$D_rCf^{(l)} = \frac{1}{2}\sum_{n=1}^{n}\sum_{j=1,i\neq j}^{n} m_{i,j^{(l)}} = (SIM_c(f_i f_j)^{(l)})^2 \tag{24}$$

The feature variables with binary mask ($m_{i,j}$) and the hyperparameter $0 \leq \tau \leq 1$ with the dot product determinant (Ω) of i and j with the regularized layers defined as:

$$
M_{i,j} = \begin{cases} 1 & \tau \leq |\Omega_{i,j}| \leq 1 \\ 0 & i = j \\ 0 & otherwise \end{cases} \tag{25}
$$

where τ is threshold parameter. The similarity that exists between two feature vectors say, f_1 and f_2, is determined by the correlation that occurs between f_1 & f_2. This correlation is defined by the cosine similarity measure (SIM_c) function as:

$$
SIM_c(f_1 f_2) = \frac{\langle f_1 f_2 \rangle}{\langle \| f_1 \| \| f_2 \| \rangle} \tag{26}
$$

The overall D_rC of the FRc was evaluated to 2.413 which has higher feature divergence capacity out of threefold measure.

4.3 | System learning approach

To enhance the classifier's accuracy and control the partial feature extraction problem, each permission type from the feature vector was learned from each the initial layer of the neural network with a learning rate of 0.01. This approach is repeated from initial layer to the final layer. At each layer, the succeeding layer learns from the preceding layer. Given the network layer $l^{\mathbb{S}} = \{1, 2 \ldots n\}$ (see Section 4), l^0 learns froml^1, l^2 learns from l^1, respectively, while updating each network. Also, where a permission feature cannot be extracted in the manifest file, zeros were paddled to form a feature vector whose attribute is not associated with permission feature vectors. This helps the classifier in detecting and differentiating permission features with a high threat to the protection level provided there is at least one (1) in the feature vector.

This learning strategy of the APC model showed a significant reduction in the time taken for the network to be trained. The approach was applied to overcome the more time it requires when using the BR regularization algorithm. Though the Levenberg-Marquardt algorithm typically requires less time, it does not result in a good generalization for difficult, small, or noisy data. Apart from enhancing classification accuracy of the model, produced a low MSE and SSE but it showed a high correlation coefficient and classification accuracy. Error estimation was significantly minimized using the approach due to the objection function in BR.

5 | EXPERIMENTAL EVALUATION

In this section, we present the performance evaluation of the APC framework. We also present experimental using the collected data set to validate the proposed permission classification model based on threat and protection level, respectively. Different experiments were conducted to determine the classifier's performance. This section also presented the experimental design, the classifier's performance metric, and comparison with state-of-art classification systems, discussion, and conclusions.

5.1 | Experimental setup

We performed the experiment in a controlled environment using a VMware workstation. The virtual environment was created and used for this research to avoid malware escalation to the outside world. The virtualization approach of the research experimental design helped in ensuring that the network infrastructure and other system resources are not affected. Our framework was implemented in python, Keras library,[65] TensorFlow,[66] and Scikit learn[67] were used for modeling our neural network, cluster modeling, and other algorithms for the classifier. We analysed the result and performance of our APC model using WEKA toolkit.[68] WEKA is an open-source Waikato environment for knowledge analysis.

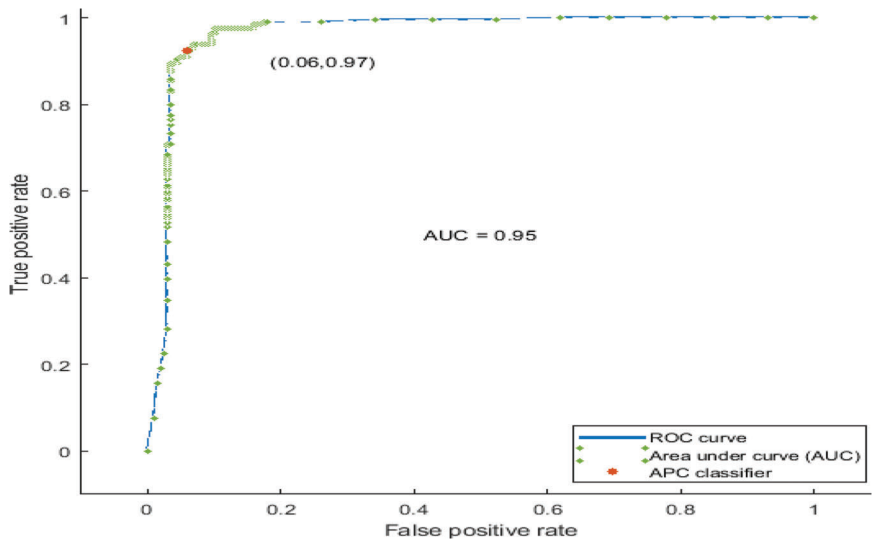


FIGURE 3 ROC showing the AUC of the classifier

TABLE 9 Performance metrics comparison with[34,71,72]

	Precision (%)	Recall (%)	FPR (%)	F_1-Score (%)	ACC (%)	AUC (%)
APC	97.0	98.60	0.20	98.0	97.00	95.00
[34]	94.30	94.50	0.56	94.60	94.38	94.60
[71]	91.60	90.70	0.86	91.80	90.72	86.00
[72]	92.40	87.50	2.38	94.12	90.54	92.00

5.2 | Evaluating performance metric

To evaluate the classifier performance, Area Under Curve (AUC) was used to measure the classifier's performance. This measurement approach helps to evaluate how efficiently the classifier can classify permission requests has been benign or malicious. Also, AUC been the area under the Receiver Operating Characteristics (ROC) curve displays the graphical representation or illustration of a binary classifier's analytical ability for its discriminate threshold (as shown in Figure 3). A different experimental test was used to derive variance measures. Precisely, our research used threshold averaging ROC curve methodology [69] rather than vertical averaging. Although vertical averaging [70] easily makes single averages of dependent variables and TPR which significantly calculates the confidence intervals and makes it simplified. However, when using a vertical averaging approach, the researcher does not have direct and absolute control of the FPR and independent variables. As a result, we used a threshold averaging curve to control the classifier's threshold scores by ensuring that at each threshold, the corresponding ROC curve was averaged.

The APC framework used the AUC also called the c-statistic of the ROC curve for performance evaluation of the APC classifier. An AUC value of >0.9 (See Figure 3) was obtained by the framework which shows outstanding discrimination of the feature variables used. These were tested centered on the utmost information gain computed, attaining 95.012% (approximately 95%) confidence interval AUC that is 0.97 ± 0.006. ROC curve and AUC represent the plot of sensitivity versus 1-Specificity as a means of efficiently determining the framework's accuracy with meaningful elucidation. ROC curve plays a fundamental function in assessing the analytical capability of the framework to differentiate the true subject states to get the best cuff of values. This helps in comparing alternative models or investigative tasks that were performed under the same subject. In summary, the ROC curve was used to evaluate and c compare the performance of the APC classifier.

5.2.1 | Comparison with state-of-art classification systems

We conducted a survey on previous works on Android permissions classifications using general machine learning algorithms, deep learning methods, and the neural networks approach. We focused our comparison on six (6) performance

TABLE 10 Performance comparison with traditional classifiers

Classifier	ACC		AUC		T_r	
	PCA_E	PCA_D	PCA_E	PCA_D	PCA_E	PCA_D
KNB	86.2	89.2	0.92	0.93	12.713	9.355
CSVM	92.5	93.7	0.91	0.92	9.989	5.067
CDT	89.9	92.2	0.90	0.93	4.204	2.0134
ABT	89.9	93.2	0.93	0.94	13.755	11.961
WKNN	90.5	93.7	0.86	0.90	3.045	2.339

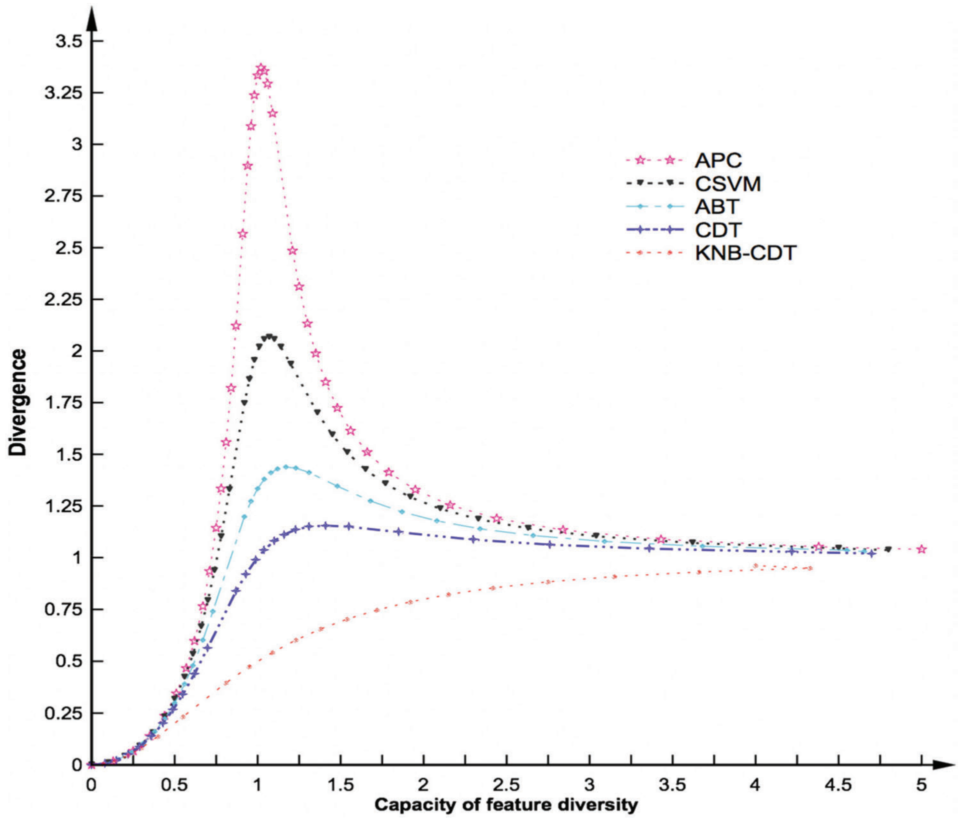


FIGURE 4 Feature diversity of the APC framework compared with other existing classifiers

metrics namely: the classifier's accuracy, precision, recall, False Positive Rate (FPR) of the classifier's value, the F-measure values, and the AUC (as shown in Table 9).

The comparative represented of the existing frameworks. APC: Our framework. The frameworks in the research of[71,34] and[72] are the previously proposed frameworks. FPR: is the false positive rate for each framework, AUC represents the AUC. F-measure is the F_1 Score which measures the test's accuracy of the binary classifiers. As shown in Table III, the machine learning detection approach proposed by Peiravian and Zhu[72] using permission and API calls obtained an accuracy value of 0.18% lower than the framework proposed by Aung and Zaw,[34] and 3.84% lower than.[71] Although all the frameworks[71,34,72] used a similar approach to feature extraction, the method proposed in Xiong et al.[71] focused on contrasting permission patterns and obtained a very low FPR compared to Aung and Zaw[34] and Peiravian and Zhu.[72] Our framework reached an FPR value of 0.20, which is significantly lower than the surveyed frameworks, and 97% classification accuracy. Finally, unlike the previous works, our framework extracts various feature vector sets categories and was not limited to opcode sequence features. By this, generalization and application of our framework when using diverse benign and malicious attributes become more effective. While the proposed approach of Bakour et al.[9] focused on opcode sequence and frequency features, our framework centralized on the feature selection principle of centroid methods.

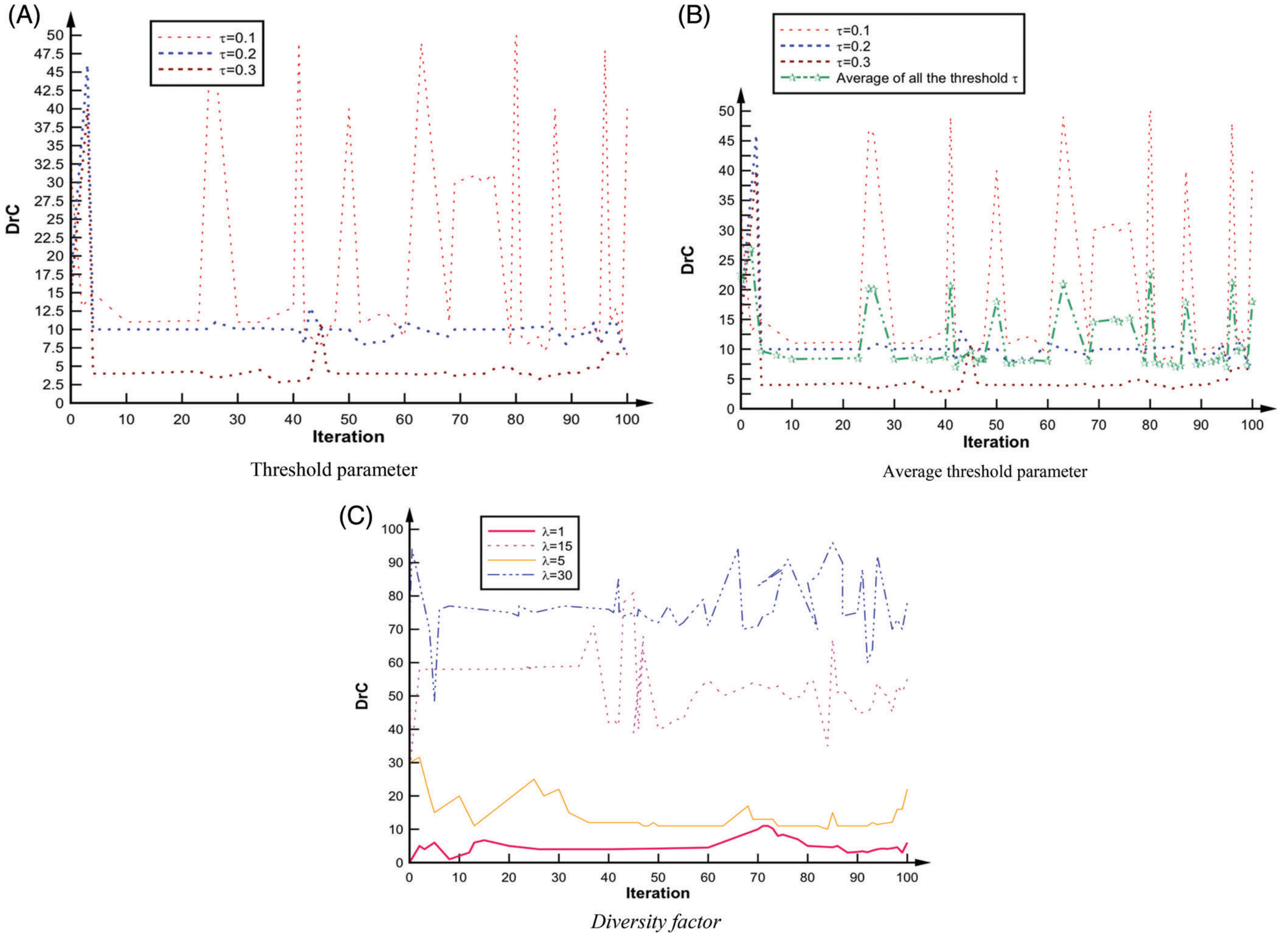


FIGURE 5 Effect of threshold and diversity factor on feature regularization

5.2.2 | Evaluating feature diversity of the APC model

We compared our framework with five traditional machine learning classifiers namely: Kernel Naïve Bayes (KNB), Cubic Support Vector Machine (KSVM), Weighted K Nearest Neighbor (WKNN), Ada Boosted Tree (ABT), and Coarse Decision Tree (CDT) to compare their capacity of feature diversity performance (as shown in Table 10). In each of the classifiers, we applied Principal Components Analysis (PCA). PCA approach enhances feature vector transformation and reduces redundancy in their dimensions. Two different experiments were conducted during PCA application on the classifiers. In the first experiment, the PCA was enabled to help the classifiers keep enough components to explain up to 95% variance. After the training, we observed that irrelevant components were kept, and only variances of the most significant components will be shown. We observed that higher accuracy was achieved when PCA was disabled than when it was disabled during the classification. In addition, more training time was required for each of the classifiers to finish classification when PCA was enabled. Less training time was taken at disabled PCA

PCA_E: Represents Principal Components Analysis enabled. PCA_D: represents Principal Components Analysis Disabled. T_r: represents the training time when PCA was both enabled and disabled. Our framework obtains a divergence value of 3.4672 and a feature diversity capacity of 4.583, respectively (as shown in Figure 4). The capacity of feature diversity of APC is higher compared to the survey classifiers in this study.

The effect of threshold and diversity factor on feature regularization (as shown in Figure 5) are determined at different threshold parameter values and iterations. The blue line from (b) indicates that the feature diversity increases as the diversity factor increases. Also, when the threshold parameter decreases, the diversity cost increases as in the yellow

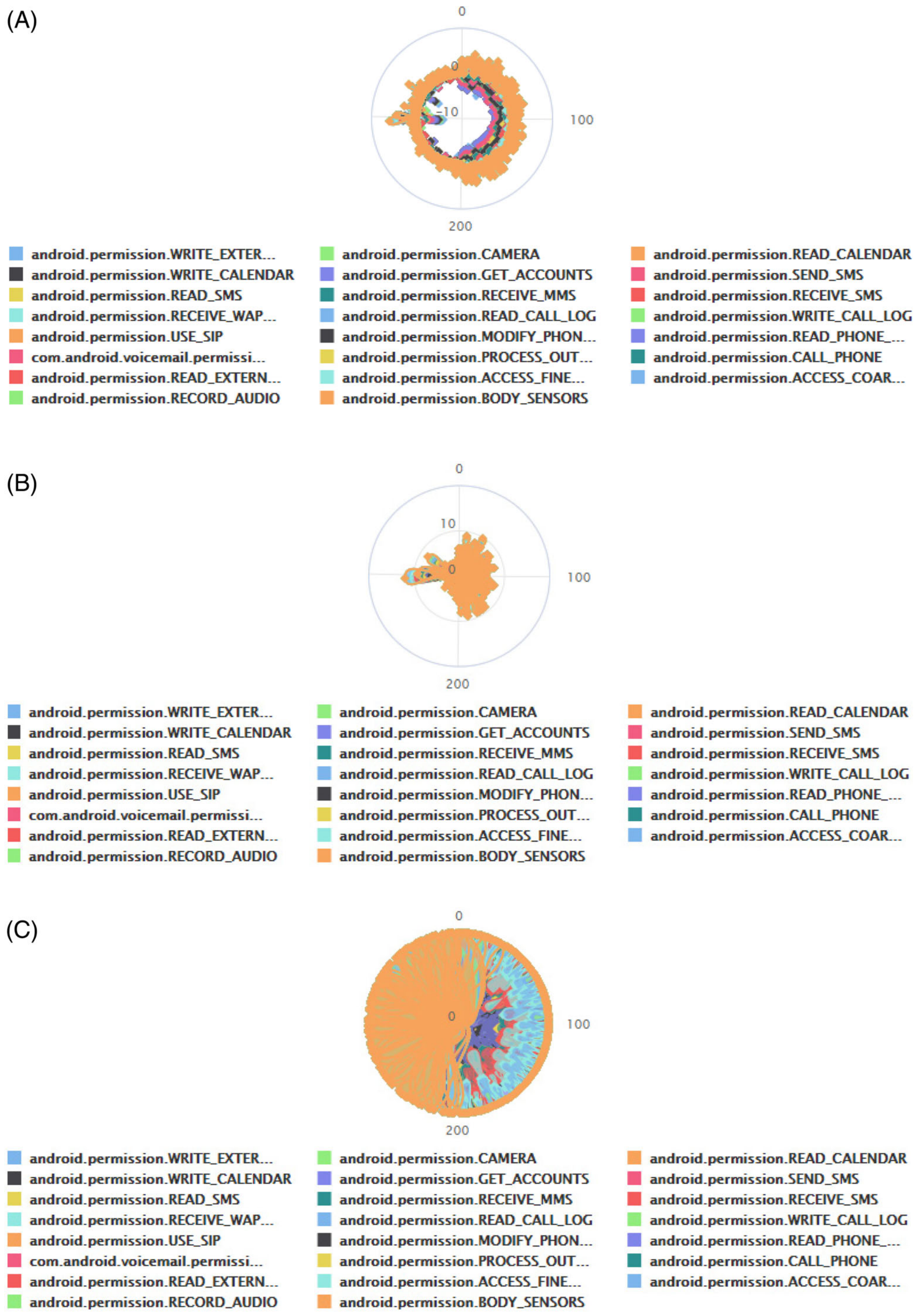


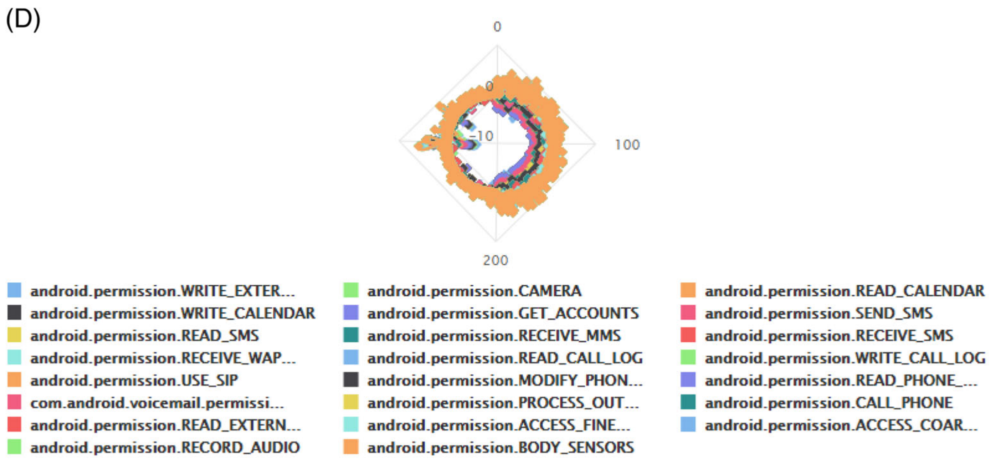
FIGURE 6 Dangerous permissions classification by the framework

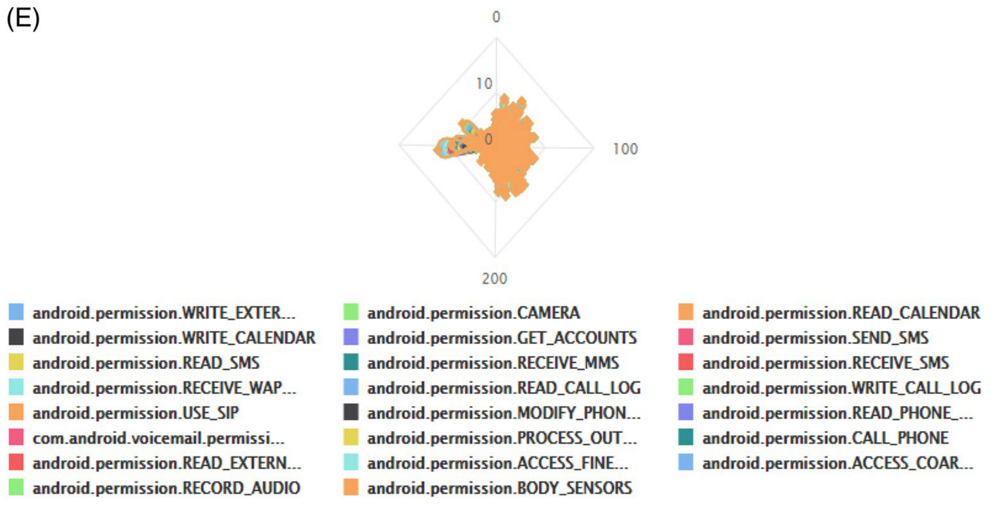
line of (a) and (c), respectively. During feature diversity experimentation, it was observed that the regularization cost for feature vector diversity increases with a decrease in threshold parameter τ and the diversity factor λ, respectively.

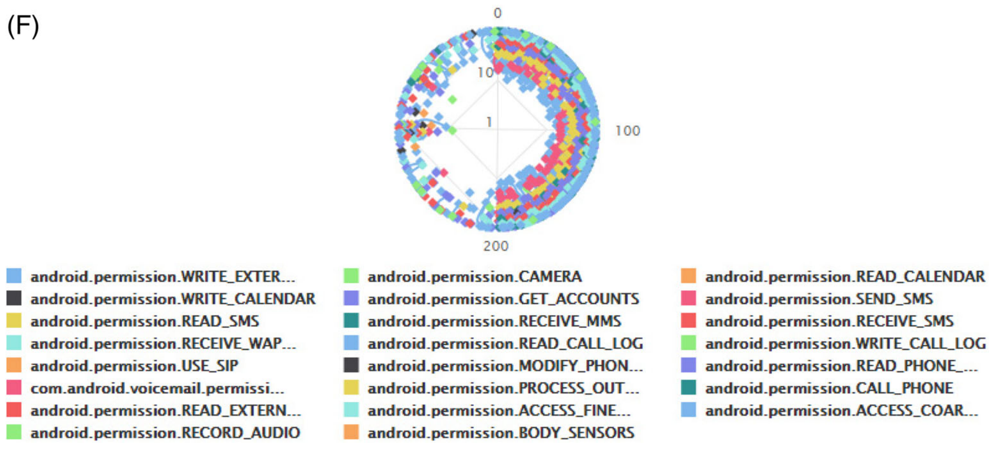
5.2.3 | Evaluating permission and threat levels

Using the association rule, the result shows that some permissions appeared frequently together. Permissions such as READ_SMS and WRITE_SMS have 97.207% of appearance. They were also frequently requested. We also found that READ_EXTERNAL_STORAGE and WRITE_EXTERNAL_STORAGE_ have a 91.503% chance to appear together. Our

FIGURE 6 (Continued)







framework consistently produces a similar pattern of the recall rates. APC classified 24 permissions as having a very high threat level to Android protection based on their frequency of occurrence and request. Permissions classification was based on polarised and radared feature vector distribution (as shown in Figure 6).

The topological data representation and analysis of dangerous features are classified. (a) are the permissions features when the feature set was polarized without stacking, (b) were classified with polarized stacking values, (c) when the training dataset values were stacked to 100 permission features, (d) when the dataset was radared without values been stacked, (e) radared with stacking values, (f) when the radared dataset stacked to 100. When the permission feature vectors

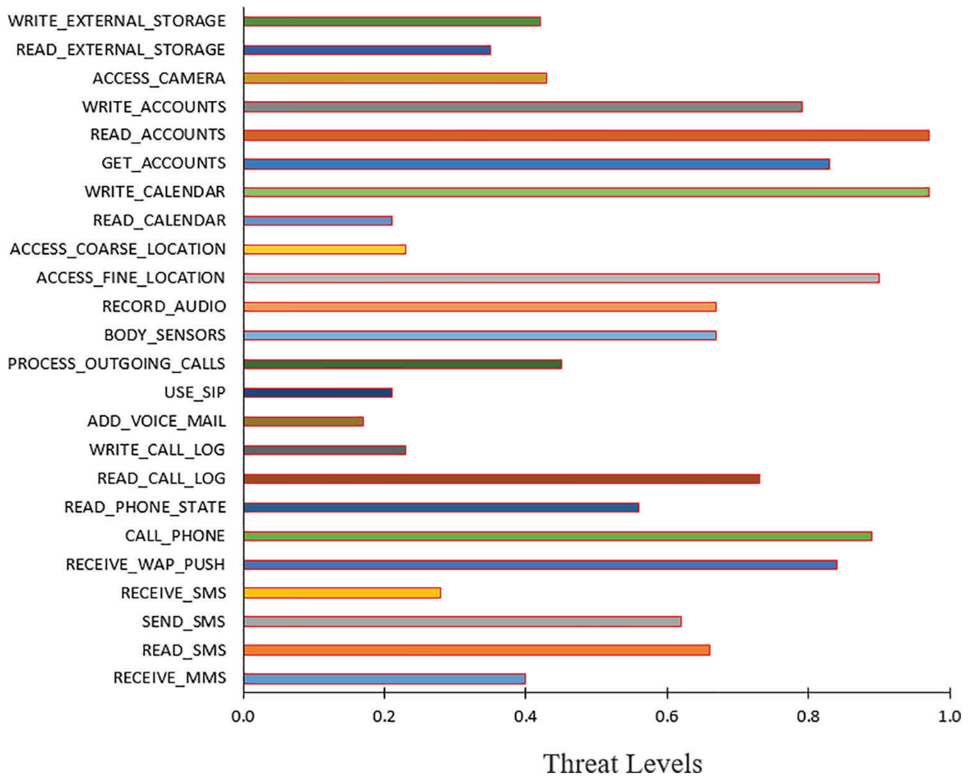


FIGURE 7 Classified permissions with different threat levels by our framework

were polarized with and without stacking by the framework, the same permission features were classified as having a high threat level to the protection level. The details of the permissions with their respective threat levels (as shown in Figure 7**).**

6 | CONCLUSION

Android is the most popular mobile OS platform in recent times. The popularity of Android brought a corresponding increase in the number of attacks targeting the OS. One of the most common and well-known attacks highly experienced by the OS is malware. Malware attacks on Android have caused several security risks ranging from device damage, privacy breach, to financial loss. Different detection and classification tools designed focus chiefly on malware classification and detection no traction to how permissions are used as attack vectors. This article aims to develop a framework for classifying android permission requests to improve the security and protection level of android mobile devices. Our research shows that the accuracy of permission requests by malicious applications can be enhanced by using an effective classification framework. Classification of android permissions with high threat levels can be improved without having much impact on the classification accuracy by using a regularization approach to increase feature diversity selection procedure. We proposed a new android permission classification framework. The best performance, obtained by combining regularization and feature diversity scheme, is a 97% classification accuracy of the model.

In general, a total of four distinct features were extracted from different APK file segments. We used feature vectors such as API calls, command signatures, permission requests to train the network. To increase the classifiers' performance run time, only 113 permissions were considered with 97% classification accuracy. This dataset is likely of Android platform version 4.4 W (KitKat Watch). Other learning algorithms tested achieved 92.4% accuracy on average when PCA was disabled and 89.8% when PCA was disabled, respectively. The results of the experiments presented in this work show a higher AUC achievement of 95% compared with the state-of-the-art permission request classification and other machine learning classifiers. We suggested a regularization function for feature diversity to reduce filtering and FRc. This helps to increase feature diversity selection procedure and generalization of the framework improvement. Our framework has a higher capacity for feature diversity when compared with traditional machine learning algorithms. Our framework classified 24 permissions as having a higher threat level to Android protection level than the rest of the permissions.

Notwithstanding the positive performance result obtained, additional research must be done to improve the accuracy and synthesis of the relationship between threat and protection level of the permission request. This forms our future work and the direction for future research.

ACKNOWLEDGMENTS

This research is supported by the Petroleum Technology Development Fund (PTDF) by a grant PTDF/ED/PHD/AMA/1245/17.

DATA AVAILABILITY STATEMENT

The data that support the findings of this study are openly available on request in the Cranfield University CORD.

ORCID

Moses Ashawa https://orcid.org/0000-0002-1016-0791

REFERENCES

1. Ashawa M, Ogwuche I. Forensic data extraction and analysis of left artifacts on emulated android phones: A case study of instant messaging applications. *Circ Comput Sci*. 2017;2(11):8-16.
2. Ashawa M, Mansour A, Haider MA. Extracting data from Android's instant messaging and social media apps. *Quar Mag Digital Forensics Pract*. 2017;33:1.
3. Smartphone Market Share. The International Data Corporation (IDC) Worldwide Quarterly Mobile Phone. https://www.idc.com/promo/smartphone-market-share/os.
4. Kim T, Kang B, Rho M, Sezer S, Im EG. A multimodal deep learning method for android malware detection using various features. *IEEE Trans Inf Forensics Secur*. 2019;14(3):773-788.
5. Tuan LH, Cam NT, Pham VH. Enhancing the accuracy of static analysis for detecting sensitive data leakage in Android by using dynamic analysis. *Clust Comput*. 2019;22(1):1079-1085.
6. Meng Z, Xiong Y, Huang W, Qin L, Jin X, Yan H. AppScalpel: Combining static analysis and outlier detection to identify and prune undesirable usage of sensitive data in Android applications. *Neurocomputing*. 2019;341:10-25.
7. Kabakus AT. What static analysis can utmost offer for android malware detection. *Inform Technand Cont*. 2019;48(2):235-240.
8. Vinayakumar R, Soman KP, Poornachandran P, Sachin Kumar SS. Detecting Android malware using long short-term memory (LSTM). *J Intell Fuzzy Syst*. 2018;34(3):1277-1288.
9. Bakour K, Ünver H, Ghanem R. The Android Malware Static analysis: Techniques, limitations, and open challenges. In 3rd International Conference on Computer Science and Engineering (UBMK) 2018, 586–593.
10. Firdaus A, Anuar NB, Karim A, Ab Razak MF. Discovering optimal features using static analysis and a genetic search-based method for Android malware detection. *Front Inf Technol Electron Eng*. 2018;19(6):712-736.
11. Li J, Sun L, Yan Q. Significant permission identification for machine-learning-based android malware detection. *IEEE Trans Ind Inform*. 2018;14(7):3216-3225.
12. Zhu HJ, You ZH, Zhu ZX, Shi Z, Chen WL, Cheng L. DroidDet: effective and robust detection of android malware using static analysis along with rotation forest model. *Neurocomputing*. 2018;272(272):638-646.
13. Lee WY, Saxe J, Harang R. SeqDroid: Obfuscated android malware detection using stacked convolutional and recurrent neural networks. *Deep learning applications for cyber security*. Cham: Springer; 2019:197-210.
14. Xiao X, Zhang S, Mercaldo F, Hu G, Sangaiah AK. Android malware detection based on system call sequences and LSTM. *Multimed Tools Appl*. 2019;78(4):3979-3999.
15. Das S, Dubrovsky AE, Korsunsky I, Dhablania A, Muender JEG. SonicWALL Inc, 2019. Just in time memory analysis for malware detection. U.S. Patent Application 15/890,192.
16. Lalande JF, Viet Triem Tong V, Graux P, Hiet G, Mazurczyk G, Berthomé P. Teaching android mobile security. Proceedings of the 50th ACM Technical Symposium on Computer Science Education, 2019, 232–238.
17. Nissim N, Lahav O, Cohen A, Elovici Y, Rokach L. Volatile memory analysis using the MinHash method for efficient and secured detection of malware in private cloud. *Comput Secur*. 2019;101590,87,1–20.
18. Grimmett Z, Staggs J, Shenoi S. Retrofitting Mobile Devices for Capturing Memory-Resident Malware Based on System Side-Effects. *IFIP Int Conf Dig Forens*. 589. Cham, Switzerland: Springer Nature Switzerland AG; 2019:59-72.
19. Zhao C, Wang C, Zheng W. Android malware detection based on sensitive permissions and APIs. Int Conf Secur Priva New Comput Environ. . Cham: Springer; 2019:105-113.
20. Kuo WC, Liu TP, Wang CC. Study on Android Hybrid Malware detection based on machine learning. 2019 IEEE 4th Int Conf Comput Commun Syst ICCCS. 2019, 31–35.
21. Wang W, Zhao M, Wang J. Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. *J Amb Intelli Hum Comput*. 2019;10(8):3035-3043.
22. Kothari S. Real time analysis of android applications by calculating risk factor to identify botnet attack. *ICCCE 2019*. Singapore: Springer; 2020:55-62.

23. Lopes J, Serrão C, Nunes L, Almeida A, Oliveira J. Overview of machine learning methods for Android malware identification. *In IEEE2019 7th Int Symp Dig Foren Secur. (ISDFS)*. 2019;1-6.
24. Fernando O, Colon C, Qiu H, Arrott A. Segmented sandboxing-A novel approach to Malware polymorphism detection. *2015 10th Int Conf Mali Unwan Soft (MALWARE)*. 2015;59-68.
25. Ashawa M, Morris S. Analysis of Android Malware Detection Techniques: A systematic review. 2019;8(3):177-187.
26. Arnatovich YL, Wang L, Ngo NM, Soh C. A comparison of android reverse engineering tools via program behaviors validation based on intermediate languages transformation. *IEEE Access*. 2018;6:12382-12394.
27. Talha KA, Alper DI, Aydin C. APK Auditor: Permission-based Android malware detection system. *Digital Investig*. 2015;13:1-14.
28. Androzoo. September 2019. [Online]. https://androzoo.uni.lu/
29. Android Malware Genome Project. Accessed September 2019. [Online]. http://www.malgenomeproject.org/
30. VirusShare. Accessed September 2019. [Online]. Available: https://virushare.com
31. Contagio Mobile. Accessed September 2019. [Online]. http://contagiomobile.deependresearch.org/index.html
32. Ashawa M, Morris S. Host-Based detection and analysis of android malware: Implication for privilege exploitation. 2019;9(2):871-880.
33. Huang C-Y, Tsai Y-T, Hsu HH. performance evaluation on permission-based detection for Android malware. *Advances in Intelligence Systems and Applications (Smart Innovation, Systems and Technologies), vol. 2*. Berlin, Germany: Springer; 2013:11-120.
34. Aung Z, Zaw W. Permission-based Android malware detection. *Int J Sci Technol Res*. 2013;2(3):228-234.
35. Arp D, Spreitzenbarth M, Hubner M, Gascon H, Rieck K. DREBIN: Effective and explainable detection of Android malware in your pocket. *Proc Netw Distrib Syst Secur Symp (NDSS)*. 2014;14:23-26.
36. Narayanan A, Yang L, Chen L, Jinliang L. Adaptive and scalable Android malware detection through Online learning. *Proc Int Joint Conf Neural Netw (IJCNN)*. 2016;2484-2491.
37. Enck W, Gilbert P, Chun BG, et al. TaintDroid: An information flow tracking system for real time privacy monitoring on smartphones. *ACM Trans Comput Syst.* 2014;32(2):5.
38. Yan LK, Yin H. DroidScope:Seamlessly reconstructing the OSand Dalvik semantic views for dynamic Android malware analysis. Proc. 21st USENIK secur. Symp. 2012, 569–584.
39. Schmidt AD, Peters F, Lamour F, Scheel C, Campete SA, Albayrak S. Monitoring smartphones for anomaly detection mobile. *Netw Appl.* 2009;14(1):92-106.
40. McLaughin N, Ricon JM, Kang B, et al.. Deep Android malware detection. Proc. ACM Conf. Data Appl. Secur. Privacy (CODASPY), 2017, 301–308.
41. Saxe J, Berlin K. Deep neural network-based malware detection using two-dimensional binary program features. Proc. 10th Int. Conf. Malicious unwanted Softw. (MALWARE), Oct. 2015, 11–20.
42. David OE, Netanyahu NS. Deep sign: Deep learning for automatic signature generation and classification. Proc. Int. Conf. Joint Conf. Neural Netw. (IJCNN), Jul. 2015, 1–8.
43. Kim T. Multimodal deep learning method for Android malware detection. *IEEE Trans Info Forensic Secur*. 2019;14(3):773-788.
44. McLaughlin N. Deep Android Malware detection. Proc. ACM Conf. Data Appl. Secur. Privacy (CODASPY), 2017, pp. 301–308.
45. Wang W, Wang X, Feng D, et al. Exploring permission-induced risk in Android applications for malicious application detection. *IEEE Trans Inform Forensics Secur*. 2015;32(2):1-14.
46. Yerima S. Android malware dataset for machine learning 2. September 2019. [Online]. https://figshare.com/articles/Android_malware_dataset_for_machine_learning_2/5854653.
47. Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps. Accessed September 2019. [Online]. https://ibotpeaches.github.io/Apktool/
48. Hao J, Zhang G, Zheng Y, Hu W, Yang K. Solution for selective harmonic elimination in asymmetric multilevel inverter based on stochastic configuration Network and Levenberg-Marquardt Algorithm. 2019 IEEE Applied Power Electronics Conference and Exposition (APEC), 2019, 2855–2858.
49. Kim B, Lim K, Cho SJ, Park M. RomaDroid: A robust and efficient technique for detecting android app clones using a tree structure and components of each app's manifest file. *IEEE Access.* 2019;7:72182-72196.
50. Jha AK, Lee S, Lee WJ. Developer mistakes in writing android manifests: An empirical study of configuration errors. *IEEE Int Work Conf Min Softw Repos.* 2017;25-36.
51. Pei W, Li J, Li H, Gao H, Wang P. ASCAA: API-level security certification of android applications. *IET Softw.* 2017;11(2):55-63.
52. Felt AP, Ha E, Egelman S, Haney A, Chin E, Wagner D. Android permissions: User attention, comprehension, and behavior. *SOUPS 2012 - Proc 8th Symp Usable Priv Secur.* 2012; 1–14.
53. Acas RK. Feature selection methods data mining to pick predictive variables. 2008.
54. Patro SGK, sahu KK. Normalization: A preprocessing stsage. *IARJSET*. 2015;2(3):20-22.
55. Firdausillah F, Mahendra DG, Zeniarja J, et al. Implementation of neural network backpropagation using audio feature extraction for classification of gamelan notes. Proc. - 2018 Int. Semin. Appl. Technol. Inf. Commun. Creat. Technol. Hum. Life, iSemantic, 2018, 570–574.
56. Lin WC, Lu YH, Tsai CF. Feature selection in single and ensemble learning-based bankruptcy prediction models. *Expert Syst*. 2019;36(1):12335.
57. Li J, Sun L, Yan Q, Li Z, Srisa-An W, Ye H. Significant Permission Identification for Machine-Learning-Based Android Malware Detection. *IEEE Trans Ind Informatics*. 2018;14(7):3216-3225.
58. Rastegari M, Ordonez V, Redmon J, Farhadi A. Xnor-net: Imagenet classification using binary convolutional neural networks. *European Conf. on Comp. Vision*. Cham: Springer; 2016:525-542.

59. Gu J, Wang Z, Kuen J, et al. Recent advances in convolutional neural networks. *Pattern Recogn.* 2018;77:354-377.
60. Al Khafaf N, El-Hag A. Bayesian regularization of neural network to predict leakage current in a salt fog environment. *IEEE Transa Dielectrics Electr Insulat.* 2018;25(2):686-693.
61. Bartilson D, Jang J, Smyth AW. Finite element model updating using objective-consistent sensitivity-based parameter clustering and Bayesian regularization. *Mech Syst Signal Proces*. 2019;114:328-345.
62. Bevan A, Kim T, Kang B, et al. A multimodal deep learning method for Android malware detection using various features. *IEEE Trans Inf Forensics Secur*. 2019;14(3):773-788.
63. Kayri M. Predictive abilities of Bayesian regularization and Levenberg–Marquardt algorithms in artificial neural networks: a comparative empirical study on social data. *Math Comput Appl.* 2016;21(2):20.
64. Ayinde BO, Inanc T, Zurada JM. Regularizing Deep Neural Networks by Enhancing Diversity in Feature Extraction. *IEEE Trans Neural Networks Learn Syst*. 2019;30(9):2650-2661.
65. Studer M, Falbel MD. Package ‘ keras’. 2019.
66. Bevan A. Machine learning with tensor flow scope: Introduction to some features of. 2017, 1–65.
67. Géron A. *Hands-on machine learning with Scikit-Learn and TensorFlow*. Boston, MA: O’Reilly Media, 2019.
68. Frank E, Hall MA, Witten IH. The WEKA workbench. *Data Min*. 2017;10:553-571.
69. Fawcett T. An introduction to ROC analysis. *Pattern Recognit Lett*. 2006;27(8):861-874.
70. Fawcett T. ROC Graphs: Notes and Practical considerations for data mining researchers ROC Graphs: Notes and practical considerations for data mining researchers. *HP Inven*. 2003;6(1):27.
71. Xiong P, Wang X, Niu W, Zhu T, Li G. Android malware detection with contrasting permission patterns. *China Commun*. 2014;11(8):1-14.
72. Peiravian N, Zhu X. Machine learning for Android malware detection using permission and API calls. Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI. 300–305.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: Ashawa M, Morris S. Android Permission Classifier: a deep learning algorithmic framework based on protection and threat levels. *Security and Privacy*. 2021;e164. https://doi.org/10.1002/spy2.164